Data Acquisition from tcpdump: Recovering and Reconstituting Clickstream Data

Charlotte Young ASEE Summer Faculty, NRL August 1, 2012
Information Management and Decision Architectures
POCs: Myriam Abramson and Steve Russell
Code 5584

Abstract—This paper investigates the feasibility of extracting clickstream data from network traces. Clickstream data capture the user Web browsing behavior and require the user's consent for the acquisition of this data directly from the browser. Extracting this data from network traces will provide another source of data for research as well as enable forensic Web analytics.

[1] Introduction

Myriam Abramson, NRL Code 5584, is the principal investigator for the project, ***Behavioral Web Analytics***. The questions she seeks to answer include:

- Can we identify individuals from their web browsing behavior given past browsing history? Why or why not?
- Can we distinguish between two web browsing histories as belonging to the same person or to different persons?

The objective of the project is to develop the necessary theoretical foundation and supporting algorithms for the detection, tracking, and prediction of Web browsing behavior leading to the identification of individuals of interest. Motivations for studying ***Behavioral Web Analytics*** include: the importance in cyberspace to link a virtual identity to a real one; the option to use behavioral web analytics to replace or complement keystroke authentication of a user; the use of behavioral web analytics to aid in online fraud detection; the fact that information dominance and hybrid wars are of significant interest to the Navy.

Three year overview of the ***Behavioral Web Analytics*** project:

Year 1. Data acquisition:
a. WebTracker plugin with user study
b. Clickstream data produced from generic tcpdump file
Year 2. Encoding and Genre Classification
Year 3. Structured Prediction

The project is in the first year, data acquisition. This paper is about the subproject, part b of year 1, "Clickstream data produced from generic tcpdumpfile." Tcpdump files can be obtained at the network level without having to install software plug-ins on the individual user's computer. This gives unlimited potential for research data as long as the information from the tcpdump file can be manipulated to look like clickstream data from individual users. Several steps have to occur before the tcpdump data is usable for this research. This was task assigned to me. The rest of this document summarizes what I did to accomplish this goal.

The problems identified in the mining of Web usage from network traces were as follows [1]:

(1) Path completion: pages are cached in the browser and might not appear as a request on the network.
(2) User identification: several users can be found behind one IP address.

Heuristics exists to get around those problems. We have identified in this project a third problem: disambiguation between machine and human behaviour. As more applications use Web services and the http protocol (e.g. Dropbox and Ajax requests), this problem will get worse in the future.

Report Documentation Page

Form Approved
OMB No. 0704-0188

1. REPORT DATE	2. REPORT TYPE	3. DATES COVERED
01 AUG 2012		**00-00-2012 to 00-00-2012**

4. TITLE AND SUBTITLE	5a. CONTRACT NUMBER
Data Acquisition from tcpdump: Recovering and Reconstituting Clickstream Data	5b. GRANT NUMBER
	5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S)	5d. PROJECT NUMBER
	5e. TASK NUMBER
	5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) **Naval Research Laboratory,Code 5584,4555 Overlook Ave., SW,Washington,DC,20375**	8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSOR/MONITOR'S ACRONYM(S)
	11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution unlimited

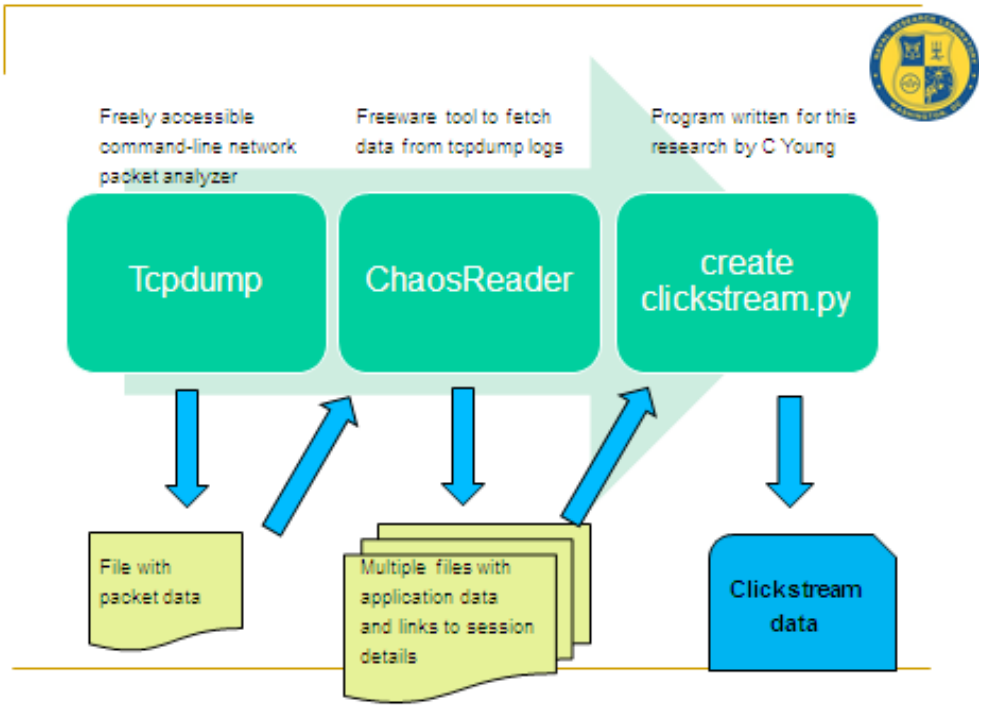
13. SUPPLEMENTARY NOTES
ASEE Summer Faculty, 2012.

14. ABSTRACT
This paper investigates the feasibility of extracting clickstream data from network traces. Clickstream data capture the user Web browsing behavior and require the user?s consent for the acquisition of this data directly from the browser. Extracting this data from network traces will provide another source of data for research as well as enable forensic Web analytics.

15. SUBJECT TERMS

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT **unclassified**	b. ABSTRACT **unclassified**	c. THIS PAGE **unclassified**	**Same as Report (SAR)**	**9**	

[2] TECHNICAL APPROACH



Some basic assumptions were made initially to constrain the scope of the project. The first was that the tcpdump files would be processed into tcp sessions using an open source program called ChaosReader. ChaosReader also reassembles the sessions into their individual application level files. As part of ChaosReader's process, it creates an index.html file that is a table of the processes sessions with links to the reassembled files. An additional assumption was made that header parsing would take a rule based approach for determining the component parts necessary to create a clickstream record.

The figure above shows the processes the data must go through to accomplish our goal of producing clickstream data from a tcpdump file. First, a tcpdump file is obtained from a network administrator. This very large file contains packet data for all network traffic during the session. Next, this tcpdump file is used as input for a freeware program called ChaosReader. This program puts the packets "back together" and produces multiple files from the information in the tcpdump files. All the files created by Chaosreader are used as input by the program written by the author, **create clickstream.py**. This Python program finds the data needed to reconstrruct single-user clickstream data. The challenges for this project include:

- Understanding all the Chaosreader files and where the desired info would come from.
- Write a program to process the data in the files and put it in clickstream form.
- Determine how to differentiate between human actions and machine generated actions (e.g. applications that would poll an html server such as inbox polling in a web based email program) in the data and then develop code to filter out the machine generated actions.

Each entry from the Chaosreader index file that makes it into the output file will have 4 important clickstream components. The components in the output file are separated by commas. First, the row number from index.html is given. This was deemed valuable for cross checking purposes during the analysis of the data. Next, the 4 clickstream components are given in this order:

Subject id	Time	URL visited	Browser agent
Use first IP addr, no :port	In ordinal format (ms)	URL address built from session file using HOST and GET or POST values. If session file not available, use second IP addr**,** **no :**port, do DNS lookup if requested	Pulled from session file

Command used to produce the tcpdump file that chaosreader then uses as input:

tcpdump -s 65535 -w out1

The decision was made to print the URL for the page visited in the reconstituted clickstream instead of its IP address that is given in the index.html file. If there is a session file for the entry, this URL can be built from finding the HOST field and concatenating the rest of the address from the GET or POST field. If there is no session file the entry is https. If requested by the user when the program is started, a lookup will be made using the IP address. The lookup for all https entries does take several seconds (or minutes) when the index.html file is large. If the lookup cannot be done, the IP address is sent to output file. The program imports "socket" module for this lookup, and then uses the following code (`IPs` is a list of ip addresses):

```
  sock = socket.getfqdn(IPs[1]) #reverse
DNS lookup for fully qualified domain name
```

It was determined that the progam must handle IP addresses in both v4 and v6. It was tricky to get the regular expression right to match IPv4 and Ipv6. The problem has to do with grouping options and how they are handled in Python's regular expressions and findall. The solution was to use raw strings and change the grouping I first thought I should use. The following code works correctly and was thoroughly tested:

```
import re

expv4 =
r"\d{1,3}\.\d{1,3}\.\d{1,3}\.\d{1,3
}|"
expv6 = r"[0-9A-Fa-f]{1,4}:[0-9A-
Fa-f]{0,4}:[0-9A-Fa-f]{0,4}:[0-9A-
Fa-f]{0,4}:[0-9A-Fa-f]{0,4}:[0-9A-
Fa-f]{0,4}:[0-9A-Fa-f]{0,4}:[0-9A-
Fa-f]{1,4}"

regexp = expv4 + expv6
```

The pattern that the program searches for in the index.html file is given by the user as a regular expression. This pattern is used to make the initial decision to pull rows from index.html as potential clickstream entries. Currently the code is written so that the search is CASE SENSITIVE. However, code is present that allows the search to be case insensitive. This code is in main and commented out but can be used instead if desired.

During the analysis of the data, the following observations were made that may be helpful in analyzing human versus machine actions:

- Google and Wikipedia send page content in compressed form (gz).
- Pages like google news have Javascripts that constantly refresh the page without user action.
- AJAX requests generate a lot of network traffic. For example, each character in a google search generates a request and gets a response so that the suggested list can be presented to the user.

An explanation from Wikipedia explains about AJAX:

> *AJAX (an acronym for Asynchronous JavaScript and XML) is a group of interrelated web development techniques used on the client-side to create asynchronous web applications. With Ajax, web applications can send data to, and retrieve data from, a server asynchronously (in the background) without interfering with the display and behavior of the existing page. Data can be retrieved using the XMLHttpRequest object. Despite the name, the use of XML is not required (JSON is often used instead), and the requests do not need to be asynchronous.*

The http standard RFC (Request For Comment) was very helpful in determining how to interpret data in the session files, which come from http packet headers. The document was used to understand GET, POST, content_type, no_cache, XMLHttpRequest , etc. that were important to the filter rubric. Web site for the RFC editor: http://www.rfc-editor.org/rfc/rfc2616.txt

This document also contained statements that are important to this project in general and should be considered during the next phase.

> 14.36 Referer The Referer[sic] request-header field allows the client to specify,for the server's benefit, the address (URI) of the resource from which the Request-URI was obtained (the "referrer", although the header field is misspelled.) The Referer request-header allows a server to generate lists of back-links to resources for interest, logging, optimized caching, etc. It also allows obsolete or mistyped links to be traced for maintenance. The Referer field MUST NOT be sent if the Request-URI was obtained from a source that does not have its own URI, such as input from the user keyboard.
>
> 15.1.1 Abuse of Server Log Information A server is in the position to save personal data about a user's requests which might identify their reading patterns or subjects of interest. This information is clearly confidential in nature and its handling can be constrained by law in certain countries. People using the HTTP protocol to provide data are responsible for ensuring that such material is not distributed without the permission of any individuals that are identifiable by the published results.
>
> 15.1.2 Transfer of Sensitive Information Like any generic data transfer protocol, HTTP cannot regulate the content of the data that is transferred, nor is there any a priori method of determining the sensitivity of any particular piece of information within the context of any given request. Therefore, applications SHOULD supply as much control over this information as possible to the provider of that information. Four header fields are worth special mention in this context: Server, Via, Referer and From. The Referer header allows reading patterns to be studied and reverse links drawn. Although it can be very useful, its power can be abused if user details are not separated from the information contained in the Referer. Even when the personal information has been removed, the Referer header might indicate a private document's URI whose publication would be inappropriate.

The actual clickstream data from the browser plugin represents only the activity of the user at his computer. The data the Python program uses is the network traffic generated by these actions, which will consist of many more entries (rows) in the ChaosReader file. The challenge is to differentiate between human action and machine generated action, filtering out the machine-generated actions and leaving only the human actions in the reconstructed clickstream.

Deciding how to filter the matched entries from the index.html file is a very important process in reconstructing the clickstream data from the original tcpdump logs. To filter out machine-generated entries it was determined that each valid entry:

- must have User Agent if it is not https
- has its content type as text/html
- will not have the field "no-cache" anywhere in the session file. (Filter out Pragma : no-cache (this is same as "Cache-Control: no-cache" and one might be present or the other.)
- will not have "0 bytes" if the pattern matches
- will not have "X-Requested-With: XMLHttpRequest" which is what will be in the header for an AJAX entry.

The list above determined the filter rubric that was decided upon. This can be applied successfully for each entry that is NOT https. When a user chooses to browse a https site, Chaosreader can provide little information. The time, userid, and site visited can be extracted, but no information for the packet headers is given. The filter rubric cannot be applied since there is no application file to open. Currently, https entries are filtered simply if there are contiguous multiple entries with the same time and URL visited, only the first of the entries is sent to the output file.

Current filter rubric:
<entries that match pattern> – (content-type != "text/html") – "no-cache"– <no browser agent if ! https> –
"X-Requested-With: XMLHttpRequest"
Next pass:
<filtered entries that are https>: only include first entry of contiguous set that have same user id, URL, and time (within ±1 second).

To figure out how to reconstruct clickstream data, several example data sets were used. Each data set included all the files generated by Chaosreader from a tcpdump of a user session. The data set also had an actual clickstream file that was generated by a plug-in to the FireFox browser used during the session. This actual clickstream could be used to compare against the reconstructed clickstream file generated by the Python program. The table on the next page gives a summary of the results using two example data sets, named in the column headers.

[3] EVALUATION

	tcpdump_path_completion	chaosreaderds 6-27
Total rows in ChaosReader index.html	1487	1030
Rows in clickstream (browser plugin)	**58**	**25**
Matches in the program, reg exp: "www|http" ***unfiltered***	444	141
Matches in the program, reg exp: "www|http" ***filtered***	**131**	**54**
(non https / https)	(61/70)	(13/41)
After "contiguous https" filter	**103**	**35**

The bottom row represents the reconstructed clickstream as produced by the python program. The goal is to have these values less than or equal to the number of rows from the actual clickstream produced by the browser plug in.

Program components that were completed:

- Parse html files and handle as text.
- Convert time "Wed May 23 11:22:33 2012" to milliseconds.
- Regular expression matching for determining file names and for matching desired rows from index.html.
- Handle both IPv4 and IPv6, with reverse lookup if page name not available.
- Taking the matched rows and filter them using rubric for determining user-generated actions.
- Create output file that looks like clickstream data.

From this point, analysis must be completed between the actual clickstream data and the program-generated clickstream data. Comparisons must be made and the filter rubric adjusted if needed. After that is complete, the program can be updated to handle data from multiple users. The path-completion problem can be researched.

To further analyze the results, a program called **_find matches.py** was written and debugged.

The purpose of this program is to analyze the filters used to see if the entries thrown out should truly be discarded. High level design for this program:

for each entry in true clickstream:
search the reconstituted clickstream for match (using URL exact match and time window in seconds)
print the true clickstream entry, print the match if found

For best analysis, this program should be executed with 3 reconstituted files:

only matches, no filters
filtered entries, no https filters
filtered entries AND https filters

[4]

Using the **tcpcump_path_completion** data set, the number of matches found is the same for both the output files **clickstream_results URL.txt** (filtered entries, no https filters) and **clickstream_results final.txt** (filtered entries AND https filters) **.** This means the https filter did not take out any matches. However, next the program should be run on completely non-filtered entries to be sure that the filters did not throw out important entries. The same process should be repeated with the other data set. This analysis should go a long way in determining how good the filters are.

[5] CONCLUSION

To summarize, it was difficult to make sense of the output of ChaosReader for our purposes. Some of the challenges included IPV6 handling, lack of handling of SSL entries and adequate handling of payload data. Another packet reassembler should be evaluated for this portion of the technical approach. A clean tcpdump/clickstream dataset should be generated by first clearing the browser of its cache. This project only identified the problem of distinguishing between machine and human Web traffic but hasn't solved it yet. Future work will attempt to address with problem with a learning approach based on http header features.

REFERENCES

[1] R. Cooley, B. Mobasher, and J. Srivastava, Data preparation for mining world wide web browsing patterns, Knowledge of Information Systems, 1999.